

Emergent Velocity Agreement in Robot Networks

Davide Canepa*

Xavier Defago[†]

Taisuke Izumi[‡]

Maria Potop-Butucaru*

Abstract

In this paper we propose and prove correct a new self-stabilizing velocity agreement (flocking) algorithm for oblivious and asynchronous robot networks. Our algorithm allows a flock of uniform robots to follow a flock head emergent during the computation whatever its direction in plane. Robots are asynchronous, oblivious and do not share a common coordinate system. Our solution includes three modules architected as follows: creation of a common coordinate system that also allows the emergence of a flock-head, setting up the flock pattern and moving the flock. The novelty of our approach steams in identifying the necessary conditions on the flock pattern placement and the velocity of the flock-head (rotation, translation or speed) that allow the flock to both follow the exact same head and to preserve the flock pattern. Additionally, our system is *self-healing* and *self-stabilizing*. In the event of the head leave (the leading robot disappears or is damaged and cannot be recognized by the other robots) the flock agrees on another head and follows the trajectory of the new head. Also, robots are *oblivious* (they do not recall the result of their previous computations) and we make no assumption on their initial position. The step complexity of our solution is $O(n)$.

*LIP6, Univ. Pierre & Marie Curie - Paris 6, LIP6-CNRS UMR 7606, France.

[†]JAIST, Japan Advanced Institute in Science and Telecommunication, Japan

[‡]Graduate School of Engineering, Nagoya Institute of Technology, Japan

1 Introduction

Flocking gained recently increased attention in diverse areas such as biology, economy, language study or agent/sensor networks. In biology, flocking refers the coordinate behaviour of a group of birds or animals when they sense some imminent threat or lookup for food. In economy, the emergent behaviour that regulates the stock markets can be seen as a form of flocking. The emergency of a common language in primitive societies is also an instantiation of flocking.

In the context of robot networks, flocking is the ability of a group of robots to coordinate and move in the plane or space. This coordinated motion has several civil and military applications ranging from zone exploration to space-crafts self-organization.

In distributed robot networks there are two types of agreement problems that have been studied so far: point or pattern agreement and velocity agreement. *Point agreement* (gathering or convergence) aims at instructing robots to reach an exact or approximate common point not known *a priori*. The dual is the scattering problem where robots are instructed to reach different positions in the plane. Furthermore, pattern agreement deals with instructing robots to eventually arrange in a predefined shape (i.e. circular, rectangular etc).

Velocity agreement or flocking refers the ability of robots to coordinate and move in 2D or 3D spaces without any external intervention. The literature agrees on two different strategies to implement flocking. The first strategy is based on a predefined hierarchy. That is, there is an *a priori* leader clearly identified in the group that will lead the group and each group member will follow the leader trajectory. An alternative is to obtain an emergent coordination of the group without a predefined leader. The difficulty of this approach comes from the permanent stress for connectivity maintenance. That is, if the flock splits then it may never converge to a single flock.

In this paper we are interested in uniform flocking strategies. That is, the head of the flock emerges during the computation hence the solution is self-healing. Also, the flock does not know *a priori* the motion trajectory. That is, the head of the flock will lead the flock following its own trajectory that may be predefined or decided on the fly. Our work is developed in the asynchronous CORDA model [5, 13] one of the two theoretical models proposed so far for oblivious distributed robot networks.

The first distributed model for robot networks, SYm, was introduced by Suzuki and Yamashita [17, 18, 19]. In SYm model robots are oblivious and perform a cycle of elementary actions as follows : observation (the robot observes the environment), computation (the robot computes its next position based on the information collected in the observation phase) and motion (the robot changes its position to the newly computed position). In this model robots cannot be interrupted during the execution of a cycle. The CORDA model breaks the execution cycle in elementary actions. That is, a robot can be activated/turned off while it executes a cycle. Hence, robots are not any more synchronized.

The *flocking problem* although largely discussed for real robots ([9, 11, 15]) was studied from distributed theoretical point of view mainly by Prencipe [7, 8]. The authors propose non-uniform algorithms where robots play one of the following roles: leader or follower. The leader is unique and all the followers know the leader robot. Obviously, when the leader crashes, disappears or duplicates the flock cannot finish its task. In [1] we extend the results in [14] and propose a probabilistic flocking architecture. However, we make the assumption that the leader and consequently the flock do not change their direction and trajectory. Our current approach is different, the leader is not known *a priori* but it will emerge during the computation. When the current leader disappears or is damaged and not recognized as a correct robot, the other robots in the system agree on another leader and the flock can finish its task. Furthermore, the flock can change both its direction and trajectory in order to agree with the emergent leader

velocity.

Fault-tolerant (but not self-stabilizing) flocking has been addressed in [16, 20]. In [16] the authors propose a fault tolerant flocking algorithm in the SYm model using a leader oracle and a failure detector. In our solution the leader or the head of the flock emerges during the computation. Also, our solution works in asynchronous settings and their decision is solely based on their current observation. In [20] the authors also propose a fault tolerant flocking. It is assumed the SYm model (awaken robots execute their operations in synchronous steps) and the k -bounded scheduler (in between two actions of a robot any other robot executes at most k actions). Also the solution needs agreement on one axis, agreement on chirality and non-oblivious robots. Contrary to this approach, our solution does not need any a priori agreement on axis or chirality. Moreover, we assume oblivious robots.

Several works from robotics propose recently heuristics for flocking (e.g. [10, 12]). In [10], for example, the authors propose a solution for non-uniform flocking. In their proposal the leader has to execute a different strategy than the rest of the flock. Hence, the system is not uniform.

Our contribution. In this paper we propose and prove correct a new asynchronous flocking algorithm in systems with oblivious and uniform robots. Additionally, we identify the necessary conditions on the flock pattern placement and the flock head velocity (rotation, translation, speed) to allow the flock to maintain the same leader and the common coordinate system and also to preserve the motion pattern. Our solution is composed of three asynchronous *self-stabilizing* and *self-healing* phases. First robots agree on a common coordinate system and a leader. Once this phase is finished robots form a flocking pattern and move preserving the same system of coordinates and the same leader. In the event of the head leave (the leading robot disappears or is damaged and cannot be recognized by the other robots) the flock agrees on another head and follows the trajectory of the new head. Also, robots are *oblivious* (they do not recall the result of their previous computations) and we make no assumption on their initial position. The complexity of our solution is $O(n)$.

Paper organization. The paper is organized as follows: Section 2 defines the model of the system, Section 3 specifies the problem based on the flocking informal definitions in different areas ranging from biological systems to space navigation. In this section we also propose a brief description of our system architecture. Section 4 sets up the common coordinate system, Section 5 details the formation of the flocking pattern and the necessary conditions on the flock placement, Section 6 proposes the rules for moving the flock and identifies the necessary conditions on the flock head velocity.

2 Model

Most of the notions presented in this section are borrowed from [8, 13]. We consider a system of autonomous mobile robots that work in the CORDA model [13].

Each robot is capable of observing its surrounding, computing a destination based on what it observed, and moving towards the computed destination: hence it performs an (endless) cycle of observing, computing, and moving. Each robot has its own local view of the world. This view includes a local Cartesian coordinate system having an origin, a unit of length, and the directions of two coordinate axis (which we will refer to as the x and y axis), together with their orientations, identified as the positive and negative sides of the axis.

The robots are model as units with computational capabilities, which are able to freely move in the plane. They are equipped with sensors that let each robot observe the positions of the

others with respect to their local coordinate system. Each robot is viewed as a point, and can see all the other robots in the flock.

The robots act totally independent and asynchronously from each other, and do not rely on any centralized directives, nor on any common notion of time. Furthermore, they are oblivious, meaning that they do not remember any previous observation nor computations performed in the previous steps. Note that this feature combined with no assumptions on the initial position of the robots gives to the algorithms designed in this model the nice property of *self-stabilization* [4]. That is, every decision taken by a robot does not depend on what happened previously in the system and robots do not use potentially corrupted data stored in their local memory.

Robots in the flock are anonymous (i.e. they are a priori indistinguishable by their appearances and they do not have any kind of identifiers that can be used during the computation). Moreover, there are no explicit direct means of communication; hence the only way they have to acquire information from their fellows robots is by observing their positions. They execute the same algorithm (the system is uniform), which takes as input the observed positions of the fellow robots, and returns a destination point towards which they target their move.

Summarizing, each robot moves totally independent and asynchronously from the others, not having any bound on the time it needs to perform a move, hence a cycle; therefore, a robot can be seen while it moves. In addition, robots are oblivious, and anonymous. We make no assumption on the initial position of robots or a common coordinate system.

3 The flocking problem

Reynolds proposed in the mid of 80's three rules that have to be respected by any algorithm that simulates a flock-like behaviour. He successfully applied these rules in designing several animations. At that time the flock entities were called boids and the model was as follows: each boid has the ability to sense its local neighbours; each boid can sense the whole environment, all boids recalculate their current state simultaneously once each time unit during the simulation.

In this model, according to Reynolds, the flocking rules are as follows:

- Separation: steer to avoid crowding local flock-mates.
- Alignment: steer towards the average heading of local flock-mates.
- Cohesion: steer to move toward the average position of local flock mates.

Interestingly, the model proposed by Reynolds is similar to the previously described SYM model. Robots can sense the environment (the other robots in the system) and they periodically and simultaneously recalculate their state. However, in CORDA model (the one used in the current work) the computation is asynchronous. Nevertheless, the main and important difference with respect to the Reynolds assumptions is the impossibility to use the history of the computation in order to implement the flocking rules. Note that the second rule of Reynolds indirectly use this information. Therefore, in the case of robot networks these rules should be adapted.

In distributed robot networks acceptance, flocking allows a group or a formation of robots to change their position either by following a pre-designated or an emergent leader. In this case the flocking is referred as *uniform*. Intuitively, a flock is a group of robots that move in the plane in order to execute a task while maintaining a specific formation. This informal definition implicitly assumes the existence of an unique leader of the flock that will lead the group during the task execution and the existence of a flocking pattern. Also it is assumed a virtual link between the head and the rest of the group. Therefore, three elements seem to be essential in the definition

of the flocking : the head or the leader of the group, the pattern and the orientation of the pattern with respect to the leader. Based on these elements, flocking can be seen as the motion of the virtual rigid formed by the flock and its head following a trajectory predefined or defined on-the-fly. It follows that both the flock and its head periodically synchronize their velocity in order to maintain the flock. In the following we specify the uniform flocking problem (i.e. the leader emerges during the computation). We first recall the definition of leader election and pattern formation. According to a recent study, [3, 6], pattern formation and leader election are related problems. Our specification naturally extends this observation to the flocking problem.

Definition 1 (Leader Election) [6] *Given the positions of n robots in the plane, the n robots are able to deterministically agree on the same robot called the leader.*

Definition 2 (Pattern Formation) [6] *The robots have in input the same pattern, called the target pattern \mathcal{F} , described as a set of positions in the plane given in lexicographic order (each robot sees the same pattern according to the direction and orientation of its local coordinate system). They are required to form the pattern: at the end of the computation, the positions of the robots coincide, in everybody's local view, with the positions of \mathcal{F} , where \mathcal{F} may be translated, rotated, and scaled in each local coordinate system.*

Definition 3 (Uniform Flocking) *Let $r_1 \dots r_n$ be a set of robots and let \mathcal{F} be the flocking pattern. The set of robots satisfy the flocking specification if the following properties hold:*

- **head/leader emergence** *eventually robots agree on an unique head (leader), r_1 ;*
- **pattern emergence** *eventually robots, r_2, \dots, r_n , form the pattern \mathcal{F} ;*
- **velocity agreement** *after any modification of the r_0 position, robots in the pattern rotate and translate \mathcal{F} in order to converge to the same relative position and orientation between r_0 and \mathcal{F} as it was before the modification.*
- **no collision** *any robot motion is collision free.*

Note the common flavour between the Reynolds rules and the above properties. *No collision* property corresponds to the separation rule. *Velocity agreement* corresponds to the alignment rule and finally *leader and pattern emergence* are similar to the cohesion property.

In the following we combine three different tasks to solve the uniform flocking in systems where robots are asynchronous, do not share the same coordinate systems, are oblivious and uniform. First, we design a novel strategy for equipping a set of robots with a common coordinate systems. To this end we propose a probabilistic strategy that creates two singularity points. Then, we combine this module with existing probabilistic election strategies ([1, 2]) in order to create the third singularity point. The motion of this third point will eventually designate the head of the flock (robot r_0) and the orientation of the common coordinate system. Then, the emergent common coordinate system is further used by all the robots but r_0 to arrange themselves in a flocking pattern, \mathcal{F} , that will further follow the head r_0 . During the pattern motion, both the head and the common coordinate system are preserved.

4 Common Coordinate System and Flock Head Emergence

The construction of a common coordinate system is as follows. First, robots agree on one axis, then they agree on the second axis, orientation of axis and the head of the flock. Due to space limitation, the details and the correctness proof of these algorithms are proposed in the Annexes (section 9).

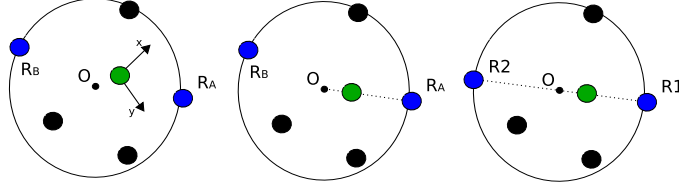


Figure 1: Alignment of *Far Robots* and Leader

Agreement on the first axis. Note that one axis is defined by two distinct points. The algorithm idea is very simple: robots compute the barycentre of their convex hull. The furthest couples of robots with respect to the barycentre (if their number is greater than one) probabilistically move further from the barycentre along the line defined by themselves and the barycentre. Two robots r_i and r_j belong to the set of the *Far Robots* if $\text{dist}(i, j) \geq \text{dist}(w, k) \forall w, k$ robots in the system. With high probability the above strategy converges to a configuration where the set *Far Robots* contains a unique couple of robots.

Agreement on the second axis. The construction of this second axis is conditioned by the existence of two unique nodes. We chose these two nodes as follows: one is the centre of smallest enclosing circle while the second one is given by any leader election algorithm. Several papers discuss the election of a leader (e.g. [1, 2]).

Axis orientation and flock head emergence. In order to orient the axis we first align the two *Far Robots*, R_A and R_B , and the leader (see Figure 1). Once the alignment is performed, the first is oriented instructing the leader to create a dissymmetry between the two points.

The alignment strategy is as follows. If the leader is not aligned with the other two robots then it will choose between the two robots belonging to the *Far Robots* the one with a bigger value of x . In case of symmetry, a bigger value of y . Then, it will move toward the intersection of the radius of that robot and the circumference of the circle with center in O (the center of the *SEC*) and the radius equal to $\text{dist}(O, \text{Leader})$. Finally, the robot not chosen by the Leader (referred as R_B) will align with the other two robots. R_B moves only when the Leader is aligned with R_A . That is, when one of the two *Far robots*, sees that the Leader is aligned with the other *Far Robot* and O , then it moves following the *SEC* until it forms a line with the Leader, the center of the *SEC* and the other robot in *Far Robots*. Note that the two *Far Robots* are on the *SEC*.

For now on, the *Far Robots* nearest to the Leader will be referred as $R1$ and the other one $R2$. Robot $R1$ will play the **flock head** role.

5 Pattern Emergence

In this section we address the formation of the flocking pattern. Note that we work in a system where robots do not have a common coordinate system. The previous section propose strategies to uniquely identify 3 robots that altogether with the center of the *SEC* define a common coordinate system. In the following, we assume that over the initial set of n robots 3 robots (referred in the previous section Leader, $R1$ and $R2$) are reserved for maintaining the coordinate system while the other $n - 3$ can be placed in any shape that will be further referred as the flocking pattern. However, we impose a condition on the placement of the shape with

respect to the position of the robots that define the references (Leader, R1 and R2) in order to preserve both the unicity of the references and the common coordinate system.

Lemma 1 *The area where the pattern is placed has to satisfy the following three conditions in order to preserve the common coordinate system defined by Leader, R1 and R2.*

1. *All robots must be inside the circumference having $\overline{R1R2}$ as diameter.*
2. *All the robots must be in the side of the SEC with R2 and y negative.*
3. *The circle with radius $\text{dist}(\text{Leader}, O)$ and center in O must be empty.*

Proof: If a robot moves outside the SEC, at the next round¹ the SEC will change and consequently the references. It follows the necessity of condition one. If there exist robots in symmetrical positions with respect to the x axis, then the system loses the capability to distinguish R1 and R2. This proves the necessity of point two. Point three is motivated by the need of an unique leader. If a robot goes closer to the center of the SEC than the leader then it becomes *Leader* and so the references will change. ■

In order to realize the flocking additional constraints on the shape of the area where the pattern is deployed are needed. These conditions will be discussed in the next section.

The flocking pattern is obtained in two steps. A first step called bootstrapping and a second step that is basically a colission free strategy to move to the pattern positions. In between these two phases the Leader will move perpendiculary to the segment $R1R2$ in order to orient the second axis. This orientation will be used by the robots in defining a total order among them.

Pattern Bootstrapping. The bootstrapping process takes two phases. In the first phase, all robots but the leader are placed on the smallest enclosing circle(SEC). First, the robots closest to the boundaries of the SEC are placed, then recursively the other robots. The algorithm avoids collisions and ensures that robots preserve the referenced (e.g. Leader, R1 and R2) computed in the previous section. In the second phase, the robots on the SEC but R1 will be placed on the semi-circle not occupied by the R1 as follows. $R1$ is in the position $SEC \cap [O, \text{Leader})$ and $R2$ is on the opposite side of the SEC. The other robots are disposed on the quarter of circle around $R2$. During this process the references are used to help the deployment of the others robots. Also, the movement of robots is done such that the semantic of the references is preserved. The code of the algorithms and their analysis are proposed in Annexes, Section 10.

Flocking Pattern Formation. In the following we defined the flocking pattern robots can form in order to maintain the common coordinate system and the references defined in the previous section.

The flocking pattern $\mathcal{F} = \{p_1, p_2, \dots, p_{n-3}, p_o, p_{R2}\}$ is the set of points given in input to the robots. It has two distinguished points p_o and p_{R2} . We call the two distinguished points the *Anchor Bolts* of the pattern, that will correspond to the position of robots $R2$ and to the point $(O, \text{dist}(\text{Leader}, O))$ of the common coordinate system. Note that robots start this phase in the following configuration: The segment $O\text{Leader}$ is perpendicular to the segment $R1R2$ (where O is the center of the SEC) and all the other robots are disposed on the quarter of circle around $R2$. In order to form the flocking pattern in a colission free manner robots adopt the following strategy. First, all robots but the references hook the pattern, eventually scale and rotate it to $R2$ and to the point $(O, \text{dist}(\text{Leader}, O))$. Then they totally order the set of

¹A round is a fragment of execution where each robot in the system executes its actions

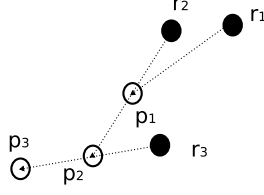


Figure 2: Pattern formation strategy

robots based on their coordinates in the common coordinate system and associate to each robot a position in the pattern. Since the order is based on the common coordinate system, all robots will define the exact same order. Then robots move following their order hence collisions are avoided. In Figure 2 robot r_2 has a trajectory that intersects the trajectory of both robots r_1 and r_3 . However collisions are avoided since following the total order r_1 moves first, then r_2 and finally r_3 . The detailed code of the algorithm and its analysis are proposed in Annexes, Section 11.

Once the flocking pattern is formed the Leader moves to the center of the *SEC*. This last movement brings the robots in what will be called latter **Flocking Formation**. The motion of this formation will be studied in the next section.

6 Velocity Agreement

In this section we propose a strategy to synchronize the *Flocking Formation* and the head of the flock. The idea is to use R_1 and R_2 as two ends of a virtual spring. The other robots will be "pulled" by R_1 and "pushed" by R_2 . Any time R_1 or R_2 moves, the center of the *SEC* changes. It follows that the *Flocking Formation* is not valid since the Leader position is not anymore on the center of the *SEC*. Then, the motion of R_1 and R_2 is blocked until the flocking pattern is reformed. The Leader moves back to the center of the *SEC* which makes the *Flocking Formation* valid and deblocks the motion of robots R_1 and R_2 .

In the following we precisely characterize two safe regions \mathcal{M} and \mathcal{K} . \mathcal{M} is the zone where R_1 is allowed to move and \mathcal{K} is the area where the pattern can be disposed. In the following we propose a general definition of \mathcal{M} and \mathcal{K} .

Definition 4 Let \mathcal{M} be the area with $y \geq \pm(kx + R_1)$. Let \mathcal{K} be the area between the axis $y = \pm h'x$ for $y < 0$ and $y = \pm hx + R_2$ for $y < 0$.

We additionally define two particular angles, α and β between the axis that border \mathcal{M} and \mathcal{K} . Latter, we prove that α and β should be greater than 90° in order to verify the conditions stated in Lemma 1.

Definition 5 Let α be the angle between $y = \pm hx + R_2$ and $y = \mp kx + R_1$. Let denote by A the intersection of these two lines. Let β be the angle between $y = \pm h'x$ and $y = \mp kx$ (see Figure 3).

The flocking algorithm, Algorithm 6.1 is executed only when the *Flocking Formation* is reached (see Section 5). The algorithm foresee the movement of robots R_1 , R_2 and the *Leader*. R_1 is the robot that imposes the direction of the movement so it can move to any point in the \mathcal{M} area. When it moves, thanks to the constraints we have imposed, the references hold steady and so, at the next observation, all the robots can recognize the references: Leader, R_1 and R_2 .

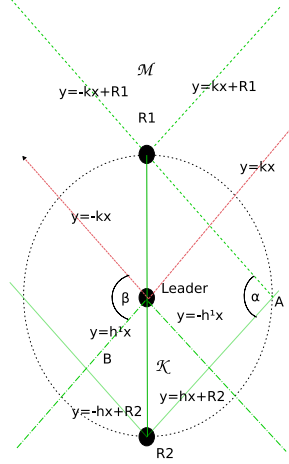


Figure 3: Angles α and β and the safe areas \mathcal{M} and \mathcal{K}

Additionally, all the other robots will be inside the *SEC* (on the $R2$ side). Then, all the other robots but $R2$ execute the flocking pattern formation algorithm (presented in Section 5) to align the pattern to the new direction (defined by the axis $R2R1$). Once the *Flocking Formation* is recreated the robots $R1$ or $R2$ can move again.

When the distance between $R1$ and $R2$ is greater than some parameter d_{Rmax} , then $R2$ moves inside the \mathcal{K} area (along the $R2R1$ segment) within distance d . Following Lemma 2 below this distance should be less or equal than $(\frac{dist(R1R2)}{2} - \frac{dist(R1B)}{2\cos\delta})$ where B is the closest robot to $R2$ and δ is the angle $\angle R2R1B$.

- 1) **if** (Robots form the **Flocking Formation**)
 - If** ($dist(R1, R2) < d_{Rmax}$)
 - then** $R1$ moves to a point $\in \mathcal{M}$;
 - 2) **else if** ($dist(R1, R2) \geq d_{Rmax}$)
 - then** $R2$ moves within distance d along the segment $R2R1$;
 - 3) **else** Leader moves perpendicular to $R1R2$ at the center of the *SEC*;
- then robots execute Flocking Formation algorithm (Section 5).

Algorithm 6.1: The motion of the Flocking Formation with parameters d and d_{Rmax} .

In the sequel we determine the relation between the axis that define the areas \mathcal{M} and \mathcal{K} (i.e. angles α and β) so that after each movement of $R1$ or $R2$ all the references are preserved. Firstly, we must guarantee that the *SEC* will change coherently with the movement of $R1$ and $R2$. At each step the *SEC* corresponds to the circumference having $R1$ and $R2$ as diameter.

Lemma 2 *Let $R1'$ be the point where robot $R1$ moves (inside the \mathcal{M} area). The circle having as diameter $R1'R2$ contains all the robots if the angle α is at least 90° .*

Proof: Consider the worst case: $R1'$ belongs to $y = \pm kx + R1$ and there exists a robot $B \neq R2$ on the border of the \mathcal{M} areas: $y = \pm hx + R2$. Without restraining the generality consider the case where $R1'$ moves on the segment $y = -kx + R1$ and B is on the line $y = hx + R2$. When $R1'$ diverges from $R1$ the circumference of the new *SEC* defined by the point $R1'$ and $R2$ intersects the line $R1B$ in a point T . When $\alpha < 90^\circ$ and $R1'$ diverges from $R1$, T moves inside the segment $R1B$ towards $R1$. Hence, at least one robot in the formation (robot B) will be

segment $\overline{R1'R2}$) which is the center of the new SEC. If $R1'$ moves on the axis $y \geq \mp kx + R1$, then O' moves on the parallel axis $y \geq \mp kx$ and if $R1'$ goes to infinity, then O' also goes to infinity.

Let B be a robot on one of the borders of \mathcal{K} , $y = \pm h'x$ for $y < 0$. In the following we determin the value of β in order to preserve the *Leader*. That is, there is no robot closer to O' than the *Leader*. Assume robot B and *Leader* are equidistant with respect to O' . Let M be the middle point of the segment $BLeader$. Notice that :

- 1) if *Leader* and B are equidistant to O' then the triangle $(B, O', Leader)$ is isosceles.
- 2) if the triangle $(B, O', Leader)$ is isosceles then the angle γ between *Leader*, M , O' is right.

It follows that $\beta < 90^0$.

So, in order to never have $\overline{O'Leader} \leq \overline{O'B}$, $\beta \geq 90^0$.

■

Lemma 5 *Starting in any configuration or after any movement of robot $R1$ or robot $R2$ the the system converges to the flocking specification in $O(n)$ steps.*

Proof: Starting in a configuration that is not a *Flocking Formation* the system converges in $O(n)$ to a *Flocking Formation*(the analysis is provided in the Annexes). After the movement of either $R1$ or $R2$, the center of the *SEC* will change. It follows that the *Leader* is not anymore on the center of the *SEC* and the *Flocking Formation* is invalide. Now, due to the first condition in Algorithm 6.1 any movement of $R1$ or $R2$ is impossible. Then the flocking formation algorithm is executed until a new *Focking Formation*, consistent with the new references, will be reached. Following the analysis provided in the Annexes this takes $O(n)$ steps. Then, R_1 and R_2 are again free to move.

■

7 Conclusions and open questions

The current work studies the flocking problem in the most generic settings: asynchronous robot networks, oblivious robots, arbitrary initial positions. We proposed a solution with nice self* properties. That is, in the event of the head leave the flock agrees on another head. Moreover, robots are oblivious, can start in any initial configuration and do not share any common knowledge. Additionally, the flock follows the head whatever its trajectory. Also, the algorithm makes sure that the flock will follow the same head (once emerged) and the system of coordinates does not change during the execution. To this end we identified the necessary conditions that both the pattern and the head velocity have to satisfy in order to maintain the flock pattern, the same unique head and the same coordinate system.

A nice extension of this problem would be to use energy constraints as in biological systems. There, in order to conserve the energy of the group, the head is replaced from time to time. Including energy considerations in the model is a challenge in itself. Also, assuming that heads may change in order to conserve the energy of the group it is also assumed that some kind of common knowledge is shared by all the members of the group. This common knowledge may help in bypassing the impossibility results related to symmetric configurations. Another interesting extension would be the volumic model. The current solution cannot work in these settings since it is based essentially on alignment properties.

Acknowledgements The last author would like to thank Ted Herman for helpfull discussions related to flocking in biological systems that inspired the current specification and also the open question stated in the Conclusion section.

References

- [1] Davide Canepa and Maria Gradinariu Potop-Butucaru. Stabilizing flocking via leader election in robot networks. In *SSS*, pages 52–66, 2007.
- [2] Yoann Dieudonne and Franck Petit. Circle formation of weak robots and lyndon words. *Inf. Process. Lett.*, 101(4):156–162, 2007.
- [3] Yoann Dieudonné, Franck Petit, and Vincent Villain. Leader election problem versus pattern formation problem. In *DISC*, pages 267–281, 2010.
- [4] Shlomi Dolev. *Self-stabilization*. MIT Press, 2000.
- [5] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. *IVS, pages 480-485.*, 2000.
- [6] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theor. Comput. Sci.*, 407(1-3):412–447, 2008.
- [7] V. Gervasi and G. Prencipe. Coordination without communication: The case of the flocking problem. *Discrete Applied Mathematics*, 2003.
- [8] Vincenzo Gervasi and Giuseppe Prencipe. Flocking by a set of autonomous mobile robots. Technical Report TR-01-24, Universitat di Pisa, 2001.
- [9] A. Qadi Jiangyang Huang, S.M. Farritor and S. Goddard. Localization and follow-the-leader control of a heterogeneous group of mobile robots. *Mechatronics, IEEE/ASME Transactions on*, 11:205–215, 2006.
- [10] Geunho Lee and Nak Young Chong. Adaptive flocking of robot swarms: Algorithms and properties. *IEICE Transactions*, 91-B(9):2848–2855, 2008.
- [11] Magnus Lindhe. *A flocking and obstacle avoidance algorithm for mobile robots*. PhD thesis, KTH Stockholm, 2004.
- [12] Christoph Moeslinger, Thomas Schmickl, and Karl Crailsheim. Emergent flocking with low-end swarm robots. In *Proceedings of the 7th international conference on Swarm intelligence*, ANTS’10, pages 424–431, 2010.
- [13] G. Prencipe. Corda: Distributed coordination of a set of autonomous mobile robots. *Proc. ERSADS, pages 185–190, May 2001.*, 2001.
- [14] Giuseppe Prencipe. Achievable patterns by an even number of autonomous mobile robots. Technical Report TR-00-11 Universitat di Pisa, 17 2000.
- [15] P. Renaud, E. Cervera, and P. Martinier. Towards a reliable vision-based mobile robot formation control. *Mechatronics, IEEE/ASME Transactions on*, 4:3176– 3181, 2004.
- [16] Samia Souissi, Taisuke Izumi, and Koichi Wada. Oracle-based flocking of mobile robots in crash-recovery model. In *SSS*, pages 683–697, 2009.
- [17] Ichiro Suzuki and Masafumi Yamashita. A theory of distributed anonymous mobile robots formation and agreement problems. Technical report, Wisconsin Univ. Milwaukee Dept. of Electrical Engineering and Computer Science, 6 1994.

- [18] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots—formation and agreement problems. *Proceedings of the 3rd International Colloquium on Structural Information and Communication Complexity (SIROCCO '96), Siena, Italy, June 1996.*, 1996.
- [19] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.
- [20] Yan Yang, Samia Souissi, Xavier Défago, and Makoto Takizawa. Fault-tolerant flocking for a group of autonomous mobile robots. *Journal of Systems and Software*, 84(1):29–36, 2011.

8 Appendix

9 Setting up a common coordinate system

The construction of a common coordinate system is built gradually. First, robots agree on one axis, then they agree on the second axis.

9.1 Agreement on the first axis

One axis is defined by two distinct points. In order to get two common points, robots move in order to distinguish an unique couple. The algorithm idea is very simple: robots compute the barycentre of their convex hull. The furthest robots with respect to the barycentre (if their number is greater than two) probabilistically moves further from the barycentre along the line defined by themselves and the barycentre.

Functions:

Far(myself) returns true if $\exists r_i, d(\text{myself}, r_i) \geq d(r_w, r_k) \forall w, k$ robots

Procedure:

FarRobots() returns the set of robots, r , $\text{Far}(r)=\text{true}$

- 1) Compute the distance $d(\text{myself}, r_i)$ between myself and each robot r_i ,
- 2) *FarRobots*()
- 3) **if** ($\text{myself} \in \mathbf{FarRobots} \wedge \|\mathbf{FarRobots}\| > 2$)
 then { compute the baricentrum of *FarRobots*()
 move away from the barycentre with probability
 $p > 0$ of a distance $d_{\text{myself}} \cdot p$ }

Algorithm 9.1: Robot Separation executed by robot *myself*

Definition 6 Two robots r_i and r_j belong to the set of the **Far Robots** if $\text{dist}(i, j) \geq \text{dist}(w, k) \forall w, k$ robots in the system.

Definition 7 (legitimate configuration) A legitimate configuration for Algorithm 9.1 is a configuration with only two **Far Robots**.

Lemma 6 Starting from a configuration with more than two robots belonging to the set **Far Robots**, the system executing Algorithm 9.1 converges to a legitimate configuration (see Definition 7) with high probability.

The proof follows the same lines as the leader election proof in [1].

9.2 Agreement on the second axis

The construction of this second axis is conditioned by the existence of two unique nodes. We chose these two nodes as follows: one is the centre of smallest enclosing circle while the second one is given by any leader election algorithm. Several papers discuss the election of a leader. In [1] the authors prove the impossibility of deterministically electing a leader without additional assumptions and propose probabilistic solutions. In [2] discuss the conditions to deterministically elect a leader.

9.3 Axis orientation and head emergence

The following strategy, referred as Algorithm 9.3 allows the alignment of points computed with Algorithm 9.1 with respect to the centre of the smallest enclosing circle and the leader. Once the alignment performed this first axis can be easily oriented using the position of the leader.

Functions:

$Lined(A, B, C)$ returns true is the three points form a line.

$Lined(A, B, C, D)$ returns true is the four points form a line.

$choose(Far Robots)$ returns one between the *Far robots*: the far robot with bigger value of y or a bigger value of x , in case y values are equal

Note: We will call the two far robots R_A and R_B only to the sake of precision in the algorithm description. While executing the algorithm robots make no distinction between them.

- 1) **if** ($R \in Far Robots$ and R not on the SEC)
 then move to the SEC .
- 2) **if** ($\neg Lined(R_A, R_B, Leader, O)$)
- 3) **if** ($\neg Lined(R_A, Leader, O)$ and $\neg Lined(R_B, Leader, O)$ and $R = Leader$)
 then move towards \overline{OX} , $X = choose(Far Robots)$ at a distance $dist(Leader, O)$.
- 4) **if** ($Lined(R_A, Leader, O)$ and $\neg Lined(R_B, Leader, O)$ and $R = R_B$)
 then move towards $\overline{R_A Leader} \cap SEC$.

In the sequel we will call $R1$ the *Far Robot* closest to *Leader* and $R2$ the other one.

Algorithm 9.2: Alignment algorithm executed by robot R

The first operation is to make sure that the two elected *Far Robots* are on the SEC . Otherwise, they move to the SEC (smallest enclosing circle).

If the leader is not aligned with the other two robots then it will choose between the two robots belonging to the *Far Robots* the one with a bigger value of x . In case of symmetry, a bigger value of y . Then, it will move toward the intersection of the radius of that robot and the circumference with center in O (the center of the SEC) and radius equals to $dist(O, Leader)$. Finally, the robot not chosen by the Leader (referred in the algorithm is R_B) will align with the other two robots. R_B moves only when the Leader is aligned with the other robot. That is, when one of the two Far robots, sees that the Leader is lined up with the other *Far robot* and O , then it moves following the SEC until it form a line with the Leader, the center of the SEC and the other robot in *Far Robots*. Recall that the two *Far robots* moved previously to the SEC .

For now on, the *Far Robots* nearest to the Leader will be referred as $R1$ (Reference 1) and the other one $R2$.

Lemma 7 *During its movement, $R2$, will never collide with another robot.*

Proof: Assume $R2$ collides with a robot $R3$. $R3$ will belong to the SEC and it will be closer than $R2$ to the point at the end of the diameter of the $R1$, so the the angle α_3 between diameter of $R1$ and $\overline{R1R3}$ will be smaller than the angle α_2 between diameter of $R1$ and $\overline{R1R2}$. But

distance(R_1, R_2) is equal to $diameter \cdot \cos(\alpha_2)$ that is smaller than $diameter \cdot \cos(\alpha_3)$. It follows that R_3 will be farther than R_2 with respect to the R_1 position which contradicts the hypothesis (R_2 is the furthest robots with respect to R_1). ■

10 Bootstrapping the flocking pattern

In this section we execute a pre-processing that prepares the set up of the flocking pattern used further by the flocking algorithm. We build on top of the algorithms described in the previous section. The pattern includes three robots used as references (called Leader, R_1 and R_2) and other $n-3$ robots that can be placed in any shape.

The bootstrapping process takes two phases. First, all robots but the leader will be placed on the smallest enclosing circle. Then, the robots on the SEC but R_1 will be placed on the semi-circle not occupied by the leader.

10.1 Phase 1: Placement on the Smallest Enclosing Cercle

The idea of the algorithm proosed as Algorithm 10.1 is very simple. First, the robots closest to the boundaries of the smallest enclosing circle are placed. Than recursively the other robots. The algorithm avoids collisions and ensures that robots preserve the referenced (e.g. Leader, R_1 and R_2) computed in the previous section.

Definition 8 *Let $FreeToMove$ be the set of robots without robots between themselves and the SEC (including the border) along the radius passing through them, and that does not belong to the SEC.*

Definition 9 *Let $AlreadyPlaced$ be the set of the robots belonging to the border of the SEC.*

Definition 10 *A legitimate configuration for Algorithm 10.1 is a configuration where all robots but the leader are in the set $AlreadyPlaced$.*

Note that the algorithm does not change the leader position neither the position of $AlreadyPlaced$ nodes (R_1 and R_2 belong to the set of the $AlreadyPlaced$). Moreover, there is no node behind the leader and sharing the same radius as the leader. Otherwise this node will be the closest to the center of the SEC.

Lemma 8 *Algorithm 10.1 is silent.*

Proof: The leader does not change its position and once all robots but the leader are in the set $AlreadyPlaced$ no robot can execute its actions so the algorithm is silent. ■

Lemma 9 *If two robots r_i and r_j belong to the set $FreeToMove$, than their final position will be different.*

Proof: If there are no robots between r_i and the SEC along $radius_i$ and there are no robots between r_j and the SEC along $radius_j$ than $radius_j$ and $radius_i$ must be different. To different radius correspond different positions on the SEC so r_i and r_j , thanks to Rule 1, will be placed on different positions. ■

Lemma 10 *A robot always moves towards a free position on the SEC.*

Preprocessing:

$\forall r_i$ compute the value of the radius passing through r_i . Let rad_{r_i} be the value of the angle between my radius ($rad_{myself} = 0$) and the radius of robot r_i , in clockwise direction (note that clockwise depends only on the coordinate system of each robot)
 $\forall r_i$ compute the value of $dist_{r_i}$, distance of the robot r_i to the border of the smallest enclosing circle (SEC)

Predicates:

$Leader(myself) \equiv \forall r_i \text{ with } i \neq myself, dist_i < dist_{myself}$

Functions:

$OccupiedPosition(rad_{myself})$: returns $r_i, i \neq myself, dist_{r_i} = 0$ and $rad_{r_i} = rad_{myself}$ otherwise \perp

$NextToMove$: returns the set of closest robots r to the SEC with $dist_r \neq 0$

```

1) if  $\neg Leader(myself) \wedge myself \in FreeToMove$ 
   then { move to  $SEC$  with distance  $dist_{myself}$  }
2) if  $(\neg Leader(myself) \wedge (myself \in NextToMove) \wedge (FreeToMove = \emptyset) \wedge$ 
    $(OccupiedPosition(rad_{myself}) \neq \perp))$ 
   then { Move to the first quarter point of the arch between robot
    $OccupiedPosition(rad_{myself})$  and robot  $r_j$  belonging to the  $SEC$  such that  $rad_j$ 
   is minimum. }

```

Algorithm 10.1: Placement executed by robot my_self

Proof: Rule 1 moves robots in the set $FreeToMove$. By definition these robots move only on free positions on the SEC .

Some robots not in the set $FreeToMove$ are allowed to move only when the set $FreeToMove$ is empty and they are in the set $NextToMove$. Let r_i and r_j be such robots. r_i and r_j are enabled for the Rule 2 and move towards the quarter point of the arc between the robot r_{ii} (and r_{jj}) and their next robot on the border of SEC . The worst case is if r_i and r_j can move together and their own system of coordinate is such that the next robot for r_i is r_{jj} and the next for r_{jj} is r_{ii} (see Figure 5).

Firstly we can notice that the arc between the two robots is free before their movement (otherwise r_j would not be the next one of r_{ii} and vice versa). Moreover, the sector between r_{jj} and r_{ii} is free. Assume a robot exists in that area. Following Rule 2 it must go towards SEC before r_i moves and hence becoming the next robot of the SEC after r_{ii} . Now r_i and r_j can move freely in this sector, the only problem can be due to a reciprocal collision. But, since r_i and r_j can move only in the first quarter of their area, starting from an opposite side (the \mathcal{K} zones of Figure 2), then they cannot collide. If instead one of the robots cannot reach the quarter point in one move (due to scheduler interference), at its next schedule it will still verify the conditions of Rule 1, so it will move to the SEC following this same rule. ■

Lemma 11 *Algorithm 10.1 is collisions free (two robots never move towards the same free position).*

Proof: Firstly if two robots r_i and r_j start at different distances from the SEC , only the one closest to SEC can move. The other one must wait until the former reaches the circumference.

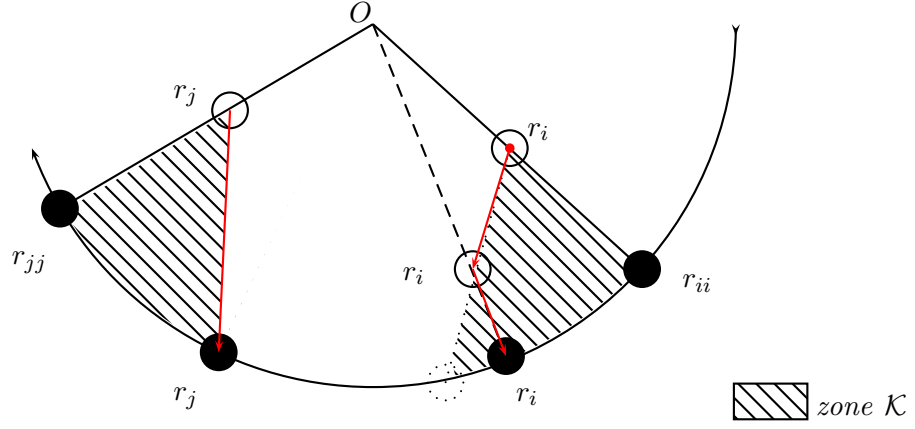


Figure 5:

If the two robots r_i and r_j have the same distance with respect to the *SEC*, than they may move at the same time. If they are enabled for the Rule 1 then they never collide since they will reach different positions on the *SEC* moving along their respective radius which are different (Lemma 9). If both r_i and r_j are enabled for Rule 2 then both robots must move towards the quarter point between the robot r_p on *SEC* belonging to its radius and the next robot r_{Pnext} on the *SEC* in their own clockwise direction.

Following Lemma 10 the sector between r_p and r_{Pnext} is free from other robots and a robot can move only inside the zone \mathcal{K} (see Figure 5). Since r_i and r_j do not belong to the same radius so $r_{p_i} \neq r_{p_j}$ and $r_{Pnext_i} \neq r_{Pnext_j}$. So the two \mathcal{K} zone where r_i and r_j can move have no intersection. It follows, r_i never reaches r_{Pnext_i} .

Overall r_i and r_j will never reach the same final position nor meet on the way to their respective final positions. ■

Lemma 12 *Algorithm 10.1 converges in $O(n)$ steps to a legitimate configuration.*

Proof: Rule 1 allows only robots in *FreeToMove* to move to the *SEC*. Thanks to Rule 2 all robots that don't belong to this set must wait until that it is empty. Once this set is empty, robots, excepted the leader, that are not on the *SEC* can execute the Rule 2. This rule is such that it can be executed only by the set of robots i with $\min(dist_i)$ and $dist_i \neq 0$. Now these robots can move towards the middle point of the arc of *SEC* between the robot on *SEC* belonging to its radius and the next robot on the *SEC* in clockwise direction. Once these robots have arrived on the *SEC* and their corresponding $dist = 0$, other robots can satisfy Rule 2 and move to the *SEC*. This process will be iterate until all the robots except the Leader are on the *SEC*. Following Lemma 11 robots do not collide and no robot obstructs the trajectory of other robots.

In the worst case the algorithm converges in $O(n)$ steps. ■

10.2 Phase 2: Setting the circular flocking configuration

This phase is a pre-processing phase for forming the final motion pattern. Starting from the final configuration of Algorithm 10.1 the curren phase reaches a circular flocking configuration as shown in Figure 6. The circular shape has the following characteristics: *Leader* is inside the *SEC* (the one computed by Algorithm 10.1) and all the other robots are disposed on the *SEC* border. These robots are placed as follows: $R1$ is in the position $SEC \cap [O, Leader)$ and $R2$ is on the opposite side of the *SEC*. The other robots are disposed on the quarter of circle around

R2. In the following this configuration will be referred as *circular flocking configuration* (see Figure 6 for a nine robots example).

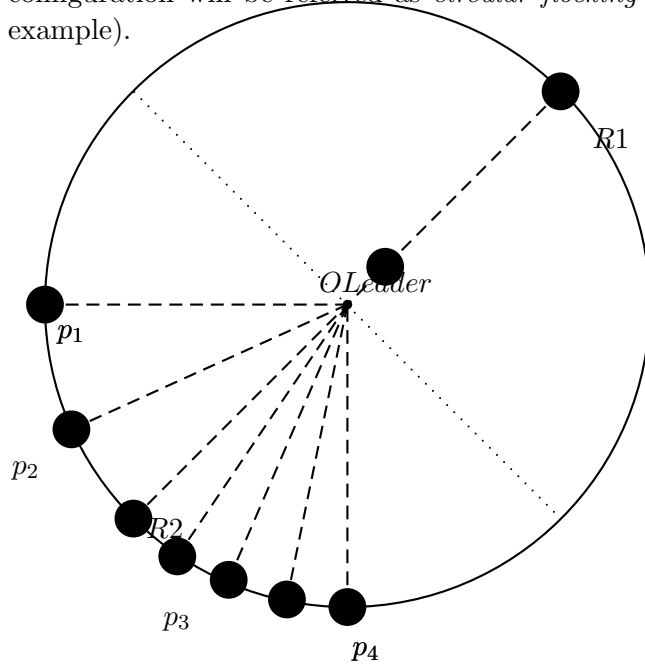


Figure 6: Circular flocking configuration

In order to construct the circular flocking configuration we introduce the concept of oriented configuration:

Definition 11 (oriented circular configuration) *A configuration is called circular oriented if the following conditions hold:*

1. All robots are at distinct positions on the SEC , except only one of them, called Leader, located inside SEC ;
2. Leader is not located at the center of SEC;
3. Two robots, named R1 and R2, form with Leader and O (the center of the SEC) a line;

Note that the output of Algorithm 10.1 satisfies point 1 of the definition above. Note also that the alignment of the references phase supply the point 2 and 3 of the definition above.

Algorithm 10.2 below started in an oriented configuration eventually converges to a circular flocking configuration.

The algorithm makes use of the following function: $FinalPositions(SEC, R1, R2)$ which returns, when invoked by a robot, the set of positions in the circular flocking configuration with respect to SEC and to the points $R1$ and $R2$. $R1$ is the robot on the intersection between the segment $[O, Leader)$ and the circle SEC . $R2$ is the robot on the intersection between the diameter of SEC passing through $R1$ and the center of SEC . To calculate the position of the other robots, we split the circumference in two parts, taking $R1$ and $R2$ as terminal points. Each robot counts how many robots are in its semi-circumference, so it will create, in the first quarter closest to $R2$, as many equidistant final positions as many robots are (considering that the last position is already occupied by $R2$). The order of these positions is given from the closest to $R1$, to the furthest

Functions:

get_number(myself) returns the number of robots between *myself* and position p_1 (including robot *myself*) clockwise
get_position(myself) returns the position *get_number(myself)* in *FinalPositions(SEC, p_1)*
FreeToMove(myself) returns true if there are no robots between *myself* and *get_position(myself)*

Motion Rule:

if FreeToMove(*myself*) **then**
 move to *get_position(myself)*

Algorithm 10.2: Setting the motion formation executed by robot *myself*

The idea of the algorithm is as follows. Robots started in an oriented configuration reach their final positions in the circular flocking configuration. If a robot is blocked by some other robots than it waits until all these robots are placed in their final positions.

Lemma 13 *In a system with n robots, Algorithm 10.2 started in an oriented configuration converges in $O(n)$ steps to a configuration where all robots reached their final positions computed via *FinalPositions* function.*

Proof: For this proof we can consider only one semi-circumference, the behavior of the other one will be totally symmetric to the former. Let p_1, \dots, p_n be the robots' final positions returned by *FinalPositions(SEC, $R1, R2$)* and let r_1, \dots, r_n be the set of robots to be placed. Assume the worst initial configuration: no robot is placed in its final position. Let denote by L the segment defined by the robots which are not placed in their final positions. The initial length of L is n . The robots r_1 (the first robot in L) can freely move to its final position p_1 . Once this robot is placed the length of segment L becomes $n - 1$. One of the new ends of L can be placed to its final position. Assume the contrary. All robots are blocked. So, there is a waiting chain such that $r_2 \rightarrow r_3 \rightarrow \dots \rightarrow r_n$ or $r_n \rightarrow \dots \rightarrow r_2$. Since the chain is finite and not cyclic (due to the total order) one of the ends of the chain can move (r_2 or r_n). After this robot moves the length of the segment L decreases. Eventually, all robots in L finish in their final positions. In the worst case, the algorithm converges in $O(n)$ steps. ■

11 Flocking Pattern

In the following we describe the pattern the robots can form.

Definition 12 (Pattern) *A pattern $P = \{p_1, p_2, \dots, p_{n-3}, p_o, p_{R2}\}$ is the set of point given in input to the robots. It has two distinguished points p_o and p_{R2} . We call the two distinguished points the Anchor Bolts of the pattern, that will correspond to the position of robots $R2$ and to the point $(O, \text{dist}(\text{Leader}, O))$ of the common coordinate system.*

First, all robots but the *references* hook the pattern, eventually scale and rotate it, to $R2$ and to the point $(O, \text{dist}(\text{Leader}, O))$. Then they associate to each robot a position in the pattern. Algorithm 11.1 ensures that each robot goes to its position without colliding with other robots. Several definitions are needed.

Definition 13 A robot r_i belongs to the set *Free Robot* if $r_i \neq p_i$.

Definition 14 A position p_i belongs to the set *Free Position* if $r_i \neq p_i$.

Functions:

Trajectory_i: the segment that joins r_i to p_i

Next(P_i, P_j): returns true if $x_{P_i} < x_{P_j}$ or, if $(x_{P_i} = x_{P_j})$ and $y_{P_i} < y_{P_j}$; otherwise returns false

We can also say P_i has P_j as Next robot(position)

Assignments to all the *Free Robots* but the references and to all *FreePosition* $\neq p_L$

but the *anchorbolts* of a sequential number

from the robot (position) with the smaller value of x to the bigger.

If two or more robots have the same x value, then consider their y value.

```

1) for (all  $r_i \neq r_{me}$ )
    If ( $p_{me} \in Trajectory_i$ )
        then Exit;
2) if (Next( $p_{me}, r_{me}$ ))
    If (Next( $r_{me-1}, p_{me}$ ))
        then (move along the  $Trajectory_{me}$  until  $x_{me-1} + \varepsilon$ );
    Else (move to  $p_{me}$ )
    If (Next( $r_{me}, p_{me}$ ))
    If (Next( $r_{me+1}, p_{me}$ ))
        then (move along the  $Trajectory_{me}$  until  $x_{me+1} - \varepsilon$ );
    Else (move to  $p_{me}$ )

```

Algorithm 11.1: Pattern Formation executed by robot "me"

Definition 15 (DeadLock) A *Deadlock configuration* is a configuration where each robot is blocked by another robot.

Note that: 1) Following Rule 1 a Deadlock arrives if there exists a robot i on each *trajectory_j*. $\forall i \neq j$ and 2) Following Rule 2 a Deadlock can occur if all the robots have a Next robot.

In the following we prove that none of these two conditions are satisfied.

Lemma 14 No deadlock configuration is reached.

Proof: In the following we prove that none of the two deadlock conditions are satisfied.

Assume that the three robots are in a DeadLock situation. We call these three robots r_1 , r_2 and r_3 . To these three robots will correspond three position p_1, p_2, p_3 . Assume also that the following statements are true: Next(p_1, p_2) and Next(p_2, p_3).

If DeadLock then: $p_1 \in Trajectory_2$ and $p_2 \in Trajectory_3$ and $p_3 \in Trajectory_1$.

In the following we prove that none of the previous situations can happen. First assume Next(r_2, p_2) returns true²: If $p_1 \in Trajectory_2$ then Next(p_1, p_2) returns true .

Let p_2 on the $Trajectory_3$ then Next(p_2, p_3) returns true. If $p_3 \in Trajectory_1$ then Next(p_3, p_1) returns true which contradicts the hypothesis Next(p_1, p_3).

²The case Next(p_2, r_2) is totally symmetric

From the definition of *Next* we can assert that the relation is unequivocal (unambiguous) so if we have two robots A and B and $\text{Next}(A,B)$ returns true then $\text{Next}(B,A)$ must return false. Also if $\text{Next}(A,B)$ returns true and $\text{Next}(B,C)$ returns true then also $\text{Next}(A,C)$ should return true. As before we can say that A and B have a next robot but C do not has any Next robot. If we iterate the process we will always find one robot without a Next robot. ■

Lemma 15 *The algorithm is collision free³.*

Proof: Easily by the point 2 no robot can surpass another robot, moreover thanks to the same point, two FreeRobots, starting from position with different x values will never occupy two positions with the same x value. Similarly if two robots start from positions with the same x value, after the activation of one of the two robots, they will be in positions with a different x value. Having at each round position with a different x value, the robots never collide. ■

Lemma 16 *All robots will eventually reach their final position in the pattern.*

Proof: Lemma 15 and Lemma 14 guarantee that all robots can move to their own position. ■

The last operation, once the pattern is bootstrapped is to bring the Leader to the center of the *SEC*. This last movement brings the robots in what will be called latter *Flocking Formation*. The motion of this formation will be studied in the next section.

³Note that is the algorithm is collision free two robots cannot try to get the same position.

